



下载APP



23 | 情感分析：如何使用LSTM进行情感分析？

2021-12-03 方远

《PyTorch深度学习实战》

课程介绍 >

**讲述：方远**

时长 11:38 大小 10.67M



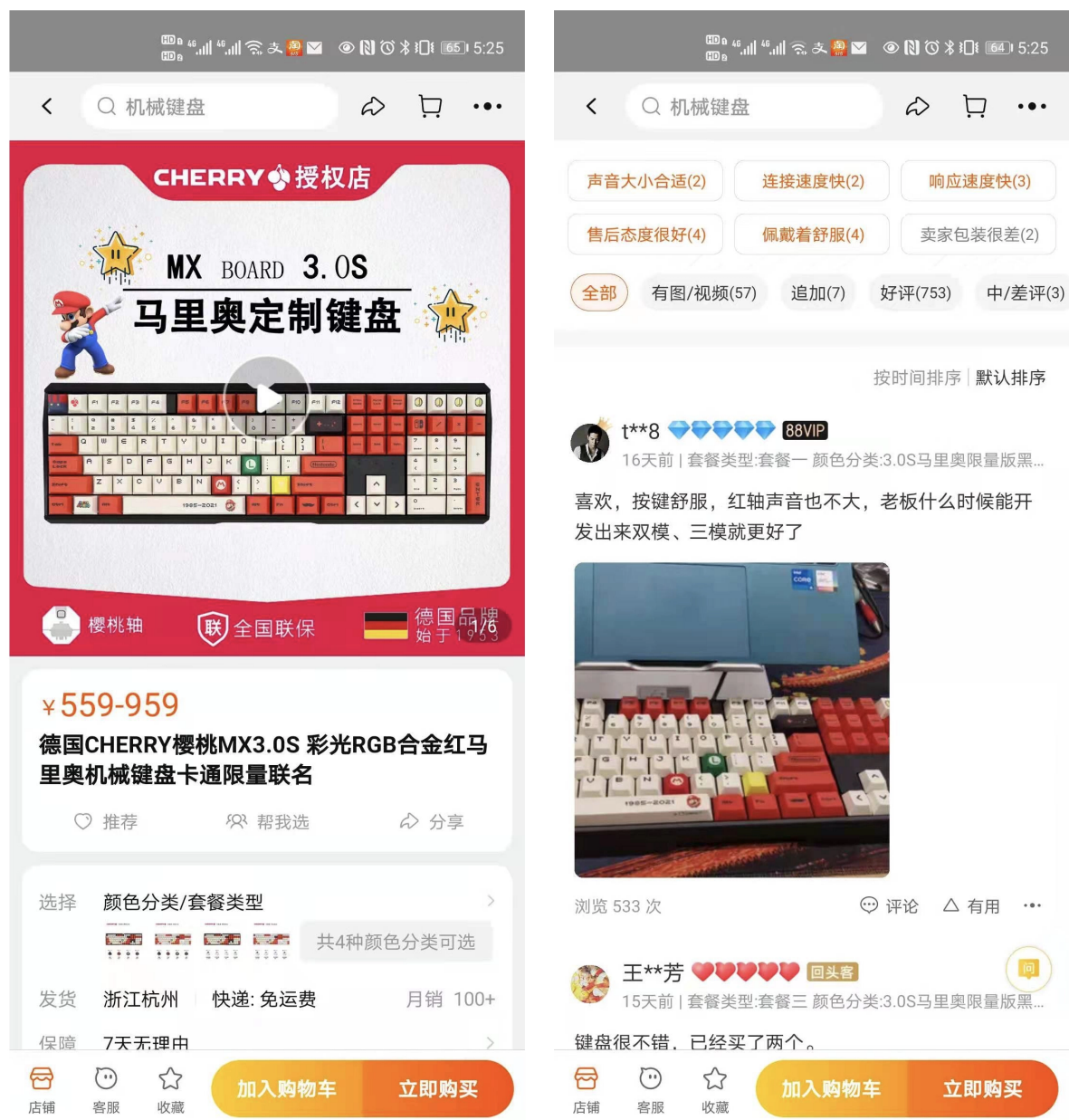
你好，我是方远。

欢迎来跟我一起学习情感分析，今天我们要讲的就是机器学习里的文本情感分析。文本情感分析又叫做观点提取、主题分析、倾向性分析等。光说概念，你可能会觉得有些抽象，我们一起来看一个生活中的应用，你一看就能明白了。

比方说我们在购物网站上选购一款商品时，首先会翻阅一下商品评价，看看是否有中差评。这些评论信息表达了人们的各种情感色彩和情感倾向性，如喜、怒、哀、乐和批评赞扬等。像这样根据评价文本，由计算机自动区分评价属于好评、中评或者说差评，用到的技术就是情感分析。



如果你进一步观察，还会发现，在好评差评的上方还有一些标签，比如“声音大小合适”、“连接速度快”、“售后态度很好”等。这些标签其实也是计算机根据文本，自动提取的主题或者观点。



情感分析的快速发展得益于社交媒体的兴起，自 2000 年初以来，情感分析已经成长为自然语言处理（NLP）中最活跃的研究领域之一，它也被广泛应用在个性化推荐、商业决策、舆情监控等方面。

今天这节课，我们将完成一个情感分析项目，一起来对影评文本做分析。

数据准备

现在我们手中有一批影评数据（IMDB 数据集），影评被分为两类：正面评价与负面评价。我们需要训练一个情感分析模型，对影评文本进行分类。

这个问题本质上还是一个文本分类问题，研究对象是电影评论类的文本，我们需要对文本进行二分类。下面我们来看一看训练数据。

IMDB（Internet Movie Database）是一个来自互联网电影数据库，其中包含了 50000 条严重两极分化的电影评论。数据集被划分为训练集和测试集，其中训练集和测试集中各有 25000 条评论，并且训练集和测试集都包含 50% 的正面评论和 50% 的消极评论。

如何用 Torchtext 读取数据集

我们可以利用 Torchtext 工具包来读取数据集。

Torchtext 是一个包含**常用的文本处理工具**和**常见自然语言数据集**的工具包。我们可以类比之前学习过的 Torchvision 包来理解它，只不过，Torchvision 包是用来处理图像的，而 Torchtext 则是用来处理文本的。

安装 Torchtext 同样很简单，我们可以使用 pip 进行安装，命令如下：

```
1 pip install torchtext
```

[复制代码](#)

Torchtext 中包含了上面我们要使用的 IMDB 数据集，并且还有读取语料库、词转词向量、词转下标、建立相应迭代器等功能，可以满足我们对文本的处理需求。

更为方便的是，Torchtext 已经把一些常见对文本处理的数据集囊括在了 torchtext.datasets 中，与 Torchvision 类似，使用时会自动下载、解压并解析数据。

以 IMDB 为例，我们可以用后面的代码来读取数据集：

```
1 # 读取IMDB数据集
2 import torchtext
```

[复制代码](#)

```
3 train_iter = torchtext.datasets.IMDB(root='./data', split='train')
4 next(train_iter)
```

`torchtext.datasets.IMDB` 函数有两个参数，其中：

`root`：是一个字符串，用于指定你想要读取目标数据集的位置，如果数据集不存在，则会自动下载；

`split`：是一个字符串或者元组，表示返回的数据集类型，是训练集、测试集或验证集，默认是（'train'，'test'）。

`torchtext.datasets.IMDB` 函数的返回值是一个迭代器，这里我们读取了 IMDB 数据集 中的训练集，共 25000 条数据，存入了变量 `train_iter` 中。

程序运行的结果如下图所示。我们可以看到，利用 `next()` 函数，读取出迭代器 `train_iter` 中的一条数据，每一行是情绪分类以及后面的评论文本。“neg”表示负面评价，“pos”表示正面评价。

```
# 读取IMDB数据集
import torchtext
train_iter = torchtext.datasets.IMDB(root='./data', split='train')
next(train_iter)

('neg',
 'I rented I AM CURIOUS-YELLOW from my video store because of all the controversy that surrounded it when it was first released in 1967. I also heard that at first it was seized by U.S. customs if it ever tried to enter this country, therefore being a fan of films considered "controversial" I really had to see this for myself.<br /><br />The plot is centered around a young Swedish drama student named Lena who wants to learn everything she can about life. In particular she wants to focus her attention on making some sort of documentary on what the average Swede thought about certain political issues such as the Vietnam War and race issues in the United States. In between asking politicians and ordinary denizens of Stockholm about their opinions on politics, she has sex with her drama teacher, classmates, and married men.<br /><br />What kills me about I AM CURIOUS-YELLOW is that 40 years ago, this was considered pornographic. Really, the sex and nudity scenes are few and far between, even then it's not shot like some cheaply made porno. While my countrymen mind find it shocking, in reality sex and nudity are a major staple in Swedish cinema. Even Ingmar Bergman, arguably their answer to good old boy John Ford, had sex scenes in his films.<br /><br />I do commend the filmmakers for the fact that any sex shown in the film is shown for artistic purposes rather than just to shock people and make money to be shown in pornographic theaters in America. I AM CURIOUS-YELLOW is a good film for anyone wanting to study the meat and potatoes (no pun intended) of Swedish cinema. But really, this film doesn't have much of a plot.')
```

数据处理 pipelines

读取出了数据集中的评论文本和情绪分类，我们还需要将文本和分类标签处理成向量，才能被计算机读取。处理文本的一般过程是先分词，然后根据词汇表将词语转换为 id。

Torchtext 为我们提供了基本的文本处理工具，包括分词器 “tokenizer” 和词汇表 “vocab”。我们可以用下面两个函数来创建分词器和词汇表。

`get_tokenizer` 函数的作用是创建一个分词器。将文本喂给相应的分词器，分词器就可以根据不同分词函数的规则完成分词。例如英文的分词器，就是简单按照空格和标点符号进行分词。

`build_vocab_from_iterator` 函数可以帮助我们使用训练数据集的迭代器构建词汇表，构建好词汇表后，输入分词后的结果，即可返回每个词语的 id。

创建分词器和构建词汇表的代码如下。首先我们要建立一个可以处理英文的分词器 `tokenizer`，然后再根据 IMDB 数据集的训练集迭代器 `train_iter` 建立词汇表 `vocab`。

[复制代码](#)

```
1 # 创建分词器
2 tokenizer = torchtext.data.utils.get_tokenizer('basic_english')
3 print(tokenizer('here is the an example!'))
4 '''
5 输出: ['here', 'is', 'the', 'an', 'example', '!']
6 '''
7
8 # 构建词汇表
9 def yield_tokens(data_iter):
10     for _, text in data_iter:
11         yield tokenizer(text)
12
13 vocab = torchtext.vocab.build_vocab_from_iterator(yield_tokens(train_iter), sp
14 vocab.set_default_index(vocab["<unk>"])
15
16 print(vocab(tokenizer('here is the an example <pad> <pad>')))
17 '''
18 输出: [131, 9, 40, 464, 0, 0]
19 '''
```

在构建词汇表的过程中，`yield_tokens` 函数的作用就是依次将训练数据集中的每一条数据都进行分词处理。另外，在构建词汇表时，用户还可以利用 `specials` 参数自定义词表。

上述代码中我们自定义了两个词语：“<pad>”和“<unk>”，分别表示占位符和未登录词。顾名思义，未登录词是指没有被收录在分词词表中的词。由于每条影评文本的长度不同，不能直接批量合成矩阵，因此需通过截断或填补占位符来固定长度。

为了方便后续调用，我们使用分词器和词汇表来建立数据处理的 pipelines。文本 pipeline 用于给定一段文本，返回分词后的 id。标签 pipeline 用于将情绪分类转化为数字，即“neg”转化为 0，“pos”转化为 1。

具体代码如下所示。

 复制代码

```
1 # 数据处理pipelines
2 text_pipeline = lambda x: vocab(tokenizer(x))
3 label_pipeline = lambda x: 1 if x == 'pos' else 0
4
5 print(text_pipeline('here is the an example'))
6 '''
7 输出: [131, 9, 40, 464, 0, 0 , ... , 0]
8 '''
9 print(label_pipeline('neg'))
10 '''
11 输出: 0
12 '''
```

通过示例的输出结果，相信你很容易就能理解文本 pipeline 和标签 pipeline 的用法了。

生成训练数据

有了数据处理的 pipelines，接下来就是生成训练数据，也就是生成 DataLoader。

这里还涉及到一个变长数据处理的问题。我们在将文本 pipeline 所生成的 id 列表转化为模型能够识别的 tensor 时，由于文本的句子是变长的，因此生成的 tensor 长度不一，无法组成矩阵。

这时，我们需要限定一个句子的最大长度。例如句子的最大长度为 256 个单词，那么超过 256 个单词的句子需要做截断处理；不足 256 个单词的句子，需要统一补位，这里用 “/” 来填补。

上面所说的这些操作，我们都可以放到 `collate_batch` 函数中来处理。

`collate_batch` 函数有什么用呢？它负责在 `DataLoad` 提取一个 batch 的样本时，完成一系列预处理工作：包括生成文本的 tensor、生成标签的 tensor、生成句子长度的 tensor，以及上面所说的对文本进行截断、补位操作。所以，我们将 `collate_batch` 函数通过参数 `collate_fn` 传入 `DataLoader`，即可实现对变长数据的处理。

`collate_batch` 函数的定义，以及生成训练与验证 `DataLoader` 的代码如下。

 复制代码

```
1 # 生成训练数据
```

```
2 import torch
3 import torchtext
4 from torch.utils.data import DataLoader
5 from torch.utils.data.dataset import random_split
6 from torchtext.data.functional import to_map_style_dataset
7
8 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
9
10 def collate_batch(batch):
11     max_length = 256
12     pad = text_pipeline('<pad>')
13     label_list, text_list, length_list = [], [], []
14     for (_label, _text) in batch:
15         label_list.append(label_pipeline(_label))
16         processed_text = text_pipeline(_text)[:max_length]
17         length_list.append(len(processed_text))
18         text_list.append((processed_text+pad*max_length)[:max_length])
19     label_list = torch.tensor(label_list, dtype=torch.int64)
20     text_list = torch.tensor(text_list, dtype=torch.int64)
21     length_list = torch.tensor(length_list, dtype=torch.int64)
22     return label_list.to(device), text_list.to(device), length_list.to(device)
23
24 train_iter = torchtext.datasets.IMDB(root='./data', split='train')
25 train_dataset = to_map_style_dataset(train_iter)
26 num_train = int(len(train_dataset) * 0.95)
27 split_train_, split_valid_ = random_split(train_dataset,
28                                           [num_train, len(train_dataset) - num_
29 train_dataloader = DataLoader(split_train_, batch_size=8, shuffle=True, collat
30 valid_dataloader = DataLoader(split_valid_, batch_size=8, shuffle=False, colla
```

我们一起梳理一下这段代码的流程，一共是五个步骤。

1. 利用 torchtext 读取 IMDB 的训练数据集，得到训练数据迭代器；
2. 使用 to_map_style_dataset 函数将迭代器转化为 Dataset 类型；
3. 使用 random_split 函数对 Dataset 进行划分，其中 95% 作为训练集，5% 作为验证集；
4. 生成训练集的 DataLoader；
5. 生成验证集的 DataLoader。

到此为止，数据部分已经全部准备完毕了，接下来我们来进行网络模型的构建。

模型构建

之前我们已经学过卷积神经网络的相关知识。卷积神经网络使用固定的大小矩阵作为输入（例如一张图片），然后输出一个固定大小的向量（例如不同类别的概率），因此适用于图像分类、目标检测、图像分割等等。

但是除了图像之外，还有很多信息，其大小或长度并不是固定的，例如音频、视频、文本等。我们想要处理这些序列相关的数据，就要用到时序模型。比如我们今天要处理的文本数据，这就涉及一种常见的时间序列模型：循环神经网络（Recurrent Neural Network, RNN）。

不过由于 RNN 自身的结构问题，在进行反向传播时，容易出现梯度消失或梯度爆炸。

LSTM 网络在 RNN 结构的基础上进行了改进，通过精妙的门控制将短时记忆与长时记忆结合起来，一定程度上解决了梯度消失与梯度爆炸的问题。

我们使用 LSTM 网络来进行情绪分类的预测。模型的定义如下。

 复制代码

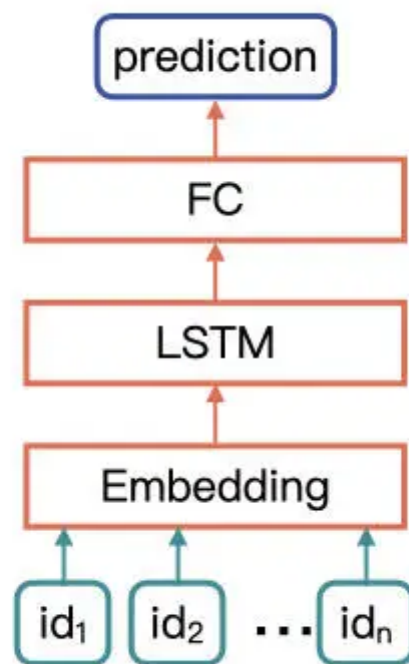
```
1 # 定义模型
2 class LSTM(torch.nn.Module):
3     def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim, n_la
4         dropout_rate, pad_index=0):
5         super().__init__()
6         self.embedding = torch.nn.Embedding(vocab_size, embedding_dim, padding
7         self.lstm = torch.nn.LSTM(embedding_dim, hidden_dim, n_layers, bidirec
8             dropout=dropout_rate, batch_first=True)
9         self.fc = torch.nn.Linear(hidden_dim * 2 if bidirectional else hidden_
10         self.dropout = torch.nn.Dropout(dropout_rate)
11
12     def forward(self, ids, length):
13         embedded = self.dropout(self.embedding(ids))
14         packed_embedded = torch.nn.utils.rnn.pack_padded_sequence(embedded, le
15             enforce_sorted=False)
16         packed_output, (hidden, cell) = self.lstm(packed_embedded)
17         output, output_length = torch.nn.utils.rnn.pad_packed_sequence(packed_
18         if self.lstm.bidirectional:
19             hidden = self.dropout(torch.cat([hidden[-1], hidden[-2]], dim=-1))
20         else:
21             hidden = self.dropout(hidden[-1])
22         prediction = self.fc(hidden)
```

23

`return prediction`

网络模型的具体结构，首先是一个 Embedding 层，用来接收文本 id 的 tensor，然后是 LSTM 层，最后是一个全连接分类层。其中，bidirectional 为 True，表示网络为双向 LSTM，bidirectional 为 False，表示网络为单向 LSTM。

网络模型的结构图如下所示。



极客时间

模型训练与评估

定义好网络模型的结构，我们就可以进行模型训练了。首先是实例化网络模型，参数以及具体的代码如下。

```
1 # 实例化模型
2 vocab_size = len(vocab)
3 embedding_dim = 300
4 hidden_dim = 300
5 output_dim = 2
6 n_layers = 2
7 bidirectional = True
8 dropout_rate = 0.5
9
10 model = LSTM(vocab_size, embedding_dim, hidden_dim, output_dim, n_layers, bidi
```

复制代码

```
11 model = model.to(device)
```

由于数据的情感极性共分为两类，因此这里我们要把 `output_dim` 的值设置为 2。

接下来是定义损失函数与优化方法，代码如下。在之前的课程里也多次讲过了，所以这里不再重复。

[复制代码](#)

```
1 # 损失函数与优化方法
2 lr = 5e-4
3 criterion = torch.nn.CrossEntropyLoss()
4 criterion = criterion.to(device)
5
6 optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```

计算 loss 的代码如下。

[复制代码](#)

```
1 import tqdm
2 import sys
3 import numpy as np
4
5 def train(dataloader, model, criterion, optimizer, device):
6     model.train()
7     epoch_losses = []
8     epoch_accs = []
9     for batch in tqdm.tqdm(dataloader, desc='training...', file=sys.stdout):
10         (label, ids, length) = batch
11         label = label.to(device)
12         ids = ids.to(device)
13         length = length.to(device)
14         prediction = model(ids, length)
15         loss = criterion(prediction, label) # loss计算
16         accuracy = get_accuracy(prediction, label)
17         # 梯度更新
18         optimizer.zero_grad()
19         loss.backward()
20         optimizer.step()
21         epoch_losses.append(loss.item())
22         epoch_accs.append(accuracy.item())
23     return epoch_losses, epoch_accs
24
25 def evaluate(dataloader, model, criterion, device):
26     model.eval()
```

```
27     epoch_losses = []
28     epoch_accs = []
29     with torch.no_grad():
30         for batch in tqdm.tqdm(dataloader, desc='evaluating...', file=sys.stdout):
31             (label, ids, length) = batch
32             label = label.to(device)
33             ids = ids.to(device)
34             length = length.to(device)
35             prediction = model(ids, length)
36             loss = criterion(prediction, label) # loss计算
37             accuracy = get_accuracy(prediction, label)
38             epoch_losses.append(loss.item())
39             epoch_accs.append(accuracy.item())
40     return epoch_losses, epoch_accs
```

可以看到，这里训练过程与验证过程的 loss 计算，分别定义在了 train 函数和 evaluate 函数中。主要区别是训练过程有梯度的更新，而验证过程中不涉及梯度的更新，只计算 loss 即可。

模型的评估我们使用 ACC，也就是准确率作为评估指标，计算 ACC 的代码如下。

[📄 复制代码](#)

```
1 def get_accuracy(prediction, label):
2     batch_size, _ = prediction.shape
3     predicted_classes = prediction.argmax(dim=-1)
4     correct_predictions = predicted_classes.eq(label).sum()
5     accuracy = correct_predictions / batch_size
6     return accuracy
```

最后，训练过程的具体代码如下。包括计算 loss 和 ACC、保存 losses 列表和保存最优模型。

[📄 复制代码](#)

```
1 n_epochs = 10
2 best_valid_loss = float('inf')
3
4 train_losses = []
5 train_accs = []
6 valid_losses = []
7 valid_accs = []
8
9 for epoch in range(n_epochs):
10     train_loss, train_acc = train(train_dataloader, model, criterion, optimize
```

```
11     valid_loss, valid_acc = evaluate(valid_dataloader, model, criterion, device)
12     train_losses.extend(train_loss)
13     train_accs.extend(train_acc)
14     valid_losses.extend(valid_loss)
15     valid_accs.extend(valid_acc)
16     epoch_train_loss = np.mean(train_loss)
17     epoch_train_acc = np.mean(train_acc)
18     epoch_valid_loss = np.mean(valid_loss)
19     epoch_valid_acc = np.mean(valid_acc)
20     if epoch_valid_loss < best_valid_loss:
21         best_valid_loss = epoch_valid_loss
22         torch.save(model.state_dict(), 'lstm.pt')
23     print(f'epoch: {epoch+1}')
24     print(f'train_loss: {epoch_train_loss:.3f}, train_acc: {epoch_train_acc:.3f}')
25     print(f'valid_loss: {epoch_valid_loss:.3f}, valid_acc: {epoch_valid_acc:.3f}')
```

我们还可以利用保存下来 `train_losses` 列表，绘制训练过程中的 loss 曲线，或使用 [第 15 课](#) 讲过的可视化工具来监控训练过程。

至此，一个完整的情感分析项目已经完成了。从数据读取到模型构建与训练的方方面面，我都手把手教给了你，希望你能以此为模板，独立解决自己的问题。

小结

恭喜你，完成了今天的学习任务。今天我们一起完成了一个情感分析项目的实践，相当于对自然语言处理任务的一个初探。我带你回顾一下今天学习的要点。

在数据准备阶段，我们可以使用 PyTorch 提供的文本处理工具包 `Torchtext`。想要掌握 `Torchtext` 也不难，我们可以类比之前详细介绍过的 `Torchvision`，不懂的地方再对应去 [查阅文档](#)，相信你一定可以做到举一反三。

模型构建时，要根据具体的问题选择适合的神经网络。卷积神经网络常被用于处理图像作为输入的预测问题；循环神经网络常被用于处理变长的、序列相关的数据。而 LSTM 相较于 RNN，能更好地解决梯度消失与梯度爆炸的问题。

在后续的课程中，我们还会讲解两大自然语言处理任务：文本分类和摘要生成，它们分别包括了判别模型和生成模型，相信那时你一定会在文本处理方面有更深层次的理解。

每课一练

利用今天训练的模型，编写一个函数 `predict_sentiment`，实现输入一句话，输出这句话的情绪类别与概率。

例如：

 复制代码

```
1 text = "This film is terrible!"
2 predict_sentiment(text, model, tokenizer, vocab, device)
3 '''
4 输出：('neg', 0.8874172568321228)
5 '''
```

欢迎你在留言区跟我交流互动，也推荐你把今天学到的内容分享给更多朋友，跟他一起学习进步。

分享给需要的人，Ta订阅后你可得 **20 元现金奖励**

 生成海报并分享

 赞 1

 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 [22 | NLP基础（下）：详解语言模型与注意力机制](#)

更多学习推荐

2021 最新大厂 Go 工程师面试真题

大厂面试真题 + 金九银十全新整理 + 核心知识全覆盖

免费领取 



精选留言 (1)

 写留言



reatiny

2021-12-04

我感觉模型部分数据流动不太明白，建议多一点解释

展开 

